

A Note on Viewing ReLU Networks as Stacked Associative Memories

Modern neural networks are usually described in terms of function approximation, optimization, and representation learning. Those lenses have been extraordinarily productive. There is, however, a complementary viewpoint that may help unify several observations:

A ReLU network can be interpreted as a stack of associative memory layers, where each layer performs data-dependent addressing followed by linear readout.

This is not a replacement for existing theory—more a reframing that may make certain behaviors easier to reason about.

The Two Faces of ReLU

The ReLU nonlinearity is typically understood through its graph: linear for positive inputs, zero otherwise. That picture is accurate—but incomplete.

There is a second, less visually obvious role:

- **Function view:** a piecewise linear activation
- **System view:** a *gate* (on/off switch)

Each ReLU unit:

- **selects** whether a feature participates in the computation
- induces a **binary decision pattern** across the layer

Taken together, a layer produces a **binary code** over its units. This code:

- partitions input space into regions
- selects a subnetwork (a subset of weights)

In effect, the layer is not just computing values—it is **routing computation**.

From Gating to Associative Memory

With this routing perspective:

- The **activation pattern** acts like an *address*
- The **weights of the next layer** act like *stored content*
- The output is a **superposition of retrieved components**

This is structurally similar to associative memory systems:

- Address → retrieve → combine

Stacking layers compounds this:

- Early layers build *coarse addressing*
- Deeper layers refine it into *highly specific addresses*

So a deep network can be seen as a **hierarchy of associative memories with progressively refined addressing schemes**.

Why This View Helps

Several empirical facts become easier to interpret:

- **Piecewise linear regions**
→ Each region corresponds to a distinct “memory access pattern”
 - **Generalization**
→ Nearby inputs share similar addresses (a form of locality sensitivity)
 - **Overparameterization**
→ Provides a large pool of reusable “memory fragments”
 - **Interference and robustness**
→ Shared features across inputs create both generalization and crosstalk
-

Locality and Hashing

The gating patterns induced by hyperplanes behave like a form of **locality-sensitive hashing (LSH)**:

- Similar inputs → similar activation patterns
- Dissimilar inputs → different patterns

Unlike classical hashing, this mapping is **learned**, not fixed.

This suggests a useful interpretation:

Training adjusts the geometry of a learned hash so that the induced “collisions” (shared representations) are beneficial rather than harmful.

A Possible Mental Block

If one focuses only on the scalar graph of ReLU, it is natural to think:

- “This is just a simple nonlinearity enabling piecewise linear approximation.”

What is easy to miss:

- The **combinatorial structure** introduced by gating
- The fact that networks are implicitly exploring a **large space of subnetworks**

This “hidden face” is less about the slope of the function and more about **which paths through the network are active**.

Directions for Further Study

Framing networks this way suggests several concrete research directions:

1. Capacity and Interference

- How many distinct “addresses” can a layer reliably support?
- When do shared features cause constructive generalization vs destructive interference?

2. Geometry of Learned Hashes

- How do learned hyperplanes distribute activation patterns?
- What makes a “good” partition of input space for downstream linear readout?

3. Conditioning of Subnetworks

- What is the effective conditioning of the linear maps induced by typical gating patterns?
- How does this relate to training stability and noise sensitivity?

4. Structured vs Random Projections

- When does structure (e.g., orthogonal or fast transforms) improve capacity or robustness?

- How close are trained layers to random feature models in practice?

5. Layerwise Methodology

- Treat each layer as:
 - an addressing mechanism (learned partition)
 - plus a memory/readout system
 - Analyze and measure them separately during training
-

On Methodology

The field has advanced rapidly through empirical success, sometimes ahead of unified theory. Certain historical moments—nonlinear separability (e.g., XOR), the effectiveness of ReLU, the role of saddle points—highlight shifts in understanding, but not always systematic reinterpretation.

A possible complementary methodology could be:

- **Explicitly separate addressing (gating) from storage (weights)**
- Measure:
 - diversity of activation patterns
 - overlap between patterns across data
 - stability of outputs under small input perturbations
- Study networks as **ensembles of linear models indexed by activation patterns**

This does not discard existing tools (optimization theory, NTK, scaling laws), but adds a structural layer that may clarify *why* they work.

Closing Thought

Neural networks are often described as function approximators. Equally, they can be viewed as:

Systems that learn how to partition input space and attach simple models to each region, reusing and recombining components across those regions.

That perspective brings them closer to associative memory, hashing, and routing systems—areas with their own mature intuitions that may still have more to offer here.
